ISIS-1106 Lenguajes y máquinas
Project 3
Fecha: May 8, 2024

The project work for the course consisted of managing and operating a language to program a robot in a two-dimensional world.

Robot Description

The robot is able to move in the world (delimited by an $n \times n$ matrix); the robot moves from cell to cell. Cells are indexed by rows and columns. The top left cell is indexed as (1,1). North is top; West is left. The robot interacts (picks and puts down) with two different types of objects (chips and balloons). Additionally, note that the robot cannot move on, or interact with obstacles in the world (gray cells).

This final project will use GOLD to perform syntactic analysis of the ROBOT programs.

Recall the definition of the language for robot programs.

A program for the robot is a sequence of definitions and instructions.

- a definition can be a variable defition or a function definiton:
  - A variable defintion is of the form: `(defvar name n)` where `name` is a variable's name and `n` is a value used to initialize the variable.
  - A function defintion is of the form: `(defun name (Params)Cs)` . This is used to define a procedure. Here `name` is the procedure name, `(Params)` is a list of parameter names for the function (separated by spaces) and `Cs` is a sequence of commands for the function.
  - `(defun name (Params)Cs)` . This is used to define a procedure. Here `name` is the procedure name, `(Params)` is a list of parameter names for the function (separated by spaces) and `Cs` is a sequence of commands for the function.
- An instruction can be a command, a control structure or a function call.
  - A command can be any one of the following:
    * `(= name n)` where `name` is a variable's name and `n` is a value. The result of this instruction is to assign the value `n` to the variable.
    * A procedure is invoked by giving its name followed by its parameter values within parenthesis as with any other instruction, for example `(funName p1 p2 p3)`.
    * `(move n)`: where `n` is a value. The robot should move `n` steps forward.
    * `(`**skip** `n)`: where `n` is a value. The robot should jump `n` steps forward.

- `(turn D)`: where `D` can be `:left`, `:right`, or `:around` (defined as constants). The robot should turn 90 degrees in the direction of the parameter in the first to cases, and 180 in the last case.
- `(face O)`: where `O` can be `:north`, `:south`, `:east`, or `:west` (all constants). The robot should turn so that it ends up facing direction `O`.
- `(`**put** `X n)`: where `X` corresponds to either `:balloons` or `:chips`, and `n` is a value. The Robot should put `n` `X`'s.
- `(`**pick** `X n)`: where `X` is either `:balloons` or `:chips`, and `n` is a value. The robot should pick `n` `X`'s.
- `(move-dir n D)`: where `n` is a value. `D` is one of `:front`, `:right`, `:left`, `:back`. The robot should move `n` positions to the front, to the left, the right or back and end up facing the same direction as it started.
- `(run-dirs Ds)`: where `Ds` is a non-empty list of directions: `:front`, `:right`, `:left`, `:back`. The robot should move in the directions indicated by the list and end up facing the same direction as it started.
- `(move-face n O)`: here `n` is a value. `O` is `:north`, `:south`, `:west`, or `:east`. The robot should face `O` and then move `n` steps.
- `(`**null**`)`: a instruction that does not do anything
- A vaue is a number, a variable, or a contant.
- These are the constants that can be used:
  - Dim : the dimensions of the board
  - myXpos: the x postition of the robot
  - myYpos: the y position of the robot
  - myChips: number of chips held by the robot
  - myBalloons: number of balloons held by the robot
  - balloonsHere: number of balloons in the robot's cell
  - ChipsHere: number of chips that can be picked
  - Spaces: number of chips that can be dropped
- A control structure can be:

  Conditional: `(`**if** `condition B1 B2)`: Executes `B1` if condition is true and `B2` if `condition` is false. B1 and B2 can be a single command or a Block

  Repeat: `(loop condition B)`: Executes `B` while `condition` is true. B can be a single command or a block.

  RepeatTimes: `(`**repeat** `n B)` where `n` is a value. `B` is executed `n` times. B is a cingle command or a block.
- A condition can be:

- `(facing? O)` where `O` is one of: north, south, east, or west
- `(blocked?)` This is true if the Robot cannot move forward.
- `(can-put? X n)` where `X` can be chips or balloons, and `n` is a value.
- `(can-pick? X n)` where `X` can be chips or balloons, and `n` is a value.
- `(can-move? D)` where `D` is one of: `:north`, `:south`, `:west`, or `:east`
- `(isZero? V)` where `V` is a value.
- `(not cond)` where `cond` is a condition.

Blocks are sequences of instructions delimited by parenthesis `()`.

Spaces, newlines, and tabulators are separators and should be ignored.

The language is not case-sensitive. This is to say, it does not distinguish between upper and lower case letters.

Remember the robot cannot walk over obstacles, and when jumping it cannot land on an obstacle. The robot cannot walk off the board or land off the board when jumping.

Task 1. The task for this project is to use GOLD to perform syntactical analysis for the Robot. Specifically, you have to complete the definition of the lexer (a finite state transducer) and the parser (a pushdown automata). You only have to determine whether a given robot program is syntactically correct: you do not have to interpret the code.

Attached you will find two GOLD projects.

LexerParserExampleMiniLisp : Here we implement the compiler of a simple language using a transducer for the lexer and a push-down automaton for the parser. In the Eclipse project, the docs folder includes a pptx file that explains how this is done.

LexerParserRobot202410 : where we have implemented a compiler for a subset of the language.

This project contains three source files:

1. `LexerParserRobot202410.gold`: the main file (the one you must run to test your program via console).
2. `Lexer202410.gold`: the lexer
3. `ParserRobot202410.gold`: the parser

The project also includes a Powerpoint presentation (doc/Ejemplo.ppt) explaining how to build a lexer and parser using GOLD.

The lexer reads the input and generates a token stream. The parser only accepts a couple of commands.

- `(move n) where n is a number`
- `(face :north)`
- `(move-dir var :right) where var is an identifier`

You have to modify `Lexer20241020.gold` so it generates tokens for the whole language. You only have to modify procedure `initialize` (line 210). You also have to modify `ParserRobot202410.gold` so you recognize the whole language.

Important: for this project, we asume that the language is case sensitive. This is to say it does distinguish between lower case and upper case letters. For example, **`moVE`** will be interpreted as an idetifier, not as the keyword **`move`** .

You do not have to verify whether or not variables and functions have been defined before they are used.

Below we show an example of a valid program.

```
(defvar rotate 3)



(if (can-move? :north ) (move-dir 1 :north) (null))

(
(if (not (blocked?)) (move 1) (null))
(turn :left)
)

(defvar one 1)

(defun foo (c p)
   (put :chips c)
   (put :balloons p)
   (move rotate))
(foo 1 3)

(defun goend ()
   (if (not (blocked?))
   ((move one)
      (goend))
   (null)))

(defun fill ()
 (repeat Spaces (if (not (isZero? myChips)) (put :chips 1) (null)
     ))
)

(defun pickAllB ()
 (pick :balloons balloonsHere)
)

(run-dirs  :left :forward :left :back :right)
```